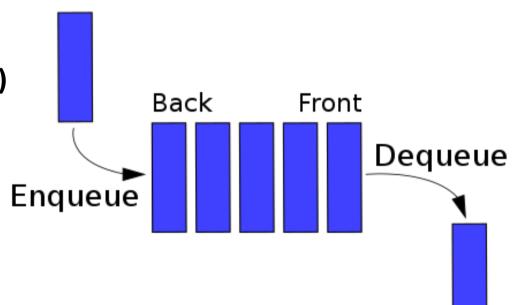
Asyncio Queue

https://github.com/hanattaw/class8-asyncio.git

Queue

- Queue is a data structure
 - items can be added by a call to put()
 - items can be retrieved by a call to get()



How to share data between coroutines using queues

- Asyncio.Queue provides a **FIFO** queue
- Asyncio.Queue cannot be used outside of an asyncio program

How to Use and Asyncio Queue

- How to create and configure an instance
- How to add items
- How to remove items
- Query the properties of the queue and manage tasks.

Create an Asyncio Queue

Queue will not be limited in capacity

```
1 ...
2 # create a queue with no size limit
3 queue = asyncio.Queue()
```

• Queue with "maxsize" set to zero (no limit) by default

```
1 ...
2 # create a queue with no size limit
3 queue = asyncio.Queue(maxsize=0)
```

Create an Asyncio Queue

Queue with limit size

```
1 ...
2 # create a queue with a size limit
3 queue = asyncio.Queue(maxsize=100)
```

 When the queue is full and coroutines attempt to add an object, they will block until space becomes available, or fail if a non-blocking method is used

Add items to Asyncio Queue

Via put() method

· Queue may block if queue is full.

```
1 ...
2 # add an object to the queue
3 await queue.put(item)
```

added to the queue without blocking via the put_nowait() method, This method is not a coroutine
and will either add the item or return immediately or fail with an asyncio.QueueFull exception if the
queue is full and the item cannot be added

```
1 ...
2 try:
3  # attempt to add an item
4  queue.put_nowait(item)
5 except asyncio.QueueFull:
6  # ...
```

Get Items from Asyncio Queue

retrieved from the queue by calling the get() method

• The item retrieved will be the oldest item added, e.g. FIFO

```
1 ...
2 # retrieve an item from the queue
3 item = await queue.get()
```

the calling coroutine may need to block until an item becomes available

Return an item immediately if available, otherwise will fail with exception

```
1 ...
2 try:
3  # attempt retrieve an item
4  item = queue.get_nowait()
5 except asyncio.QueueEmpty:
6  # ...
```

Query Asyncio Queue size

Queue size

```
1 ...
2 # report the size of the queue
3 print(queue.maxsize)
```

• Check the queue is empty via the empty()

```
1 ...
2 # check if the queue is empty
3 if queue.empty():
4 # ...
```

• Check queue is full via full()

```
1 ...
2 # check if the queue is full
3 if queue.full():
4 # ...
```

Asyncio Queue Join and Task done

- Item on the queue can be treated as tasks that can be marked as processes by consumer coroutines
- Once processed marking them via the task_done()

```
1 ...
2 # retrieve an item from the queue
3 item = await queue.get()
4 # process the item
5 # ...
6 # mark the item as processes
7 queue.task_done()
```

Asyncio Queue Join and Task done

- Other coroutines may be interest to know when all items added to the queue have been retrieved and marked as done
- This can be achieved by the coroutine awaiting the join()
- The join() coroutine will not return until all items added to the queue prior to the call have been marked as done

```
1 ...
2 # wait for all items on the queue to be marked as done
3 await queue.join()
```

If the queue is empty or all items have already been marked as done, then join() will return immediately

Example of Asyncio Queue

- we will create a producer coroutine that will generate ten random numbers and put them on the queue. We will also create a consumer coroutine that will get numbers from the queue and report their values.
- The asyncio.Queue provides a way to allow these producer and consumer coroutines to communicate data with each other.

```
    1-queue-answer.py > 分 producer

      # we will create a producer coroutine that will generate ten random numbers
      # and put them on the queue. We will also create a consumer coroutine
      # that will get numbers from the queue and report their values.
  3
  5
      from random import random
                                                                            # coroutine to consume work
                                                                      24
      import asyncio
                                                                      25 \sim async def consumer(queue):
      import time
                                                                                print(f'{time.ctime()} Consumer: Running')
                                                                      26
  8
                                                                                # consume work
                                                                      27
 9
      # coroutine to generate work
                                                                                                                                     Producer: Running
                                                                                while True:
                                                                      28 🗸
10
      async def producer(queue):
                                                                                                                                     Consumer: Running
                                                                                                                                     >got 0.7559246569022605
                                                                      29
                                                                                     # get a unit of work
          print(f'{time.ctime()} Producer: Running')
11
                                                                                                                                   4 >got 0.965203750033905
                                                                                    item = await queue.get()
                                                                      30
                                                                                                                                   5 >got 0.49834912260024233
12
           # generate work
                                                                                                                                   6 > aot 0.22783211775499135
                                                                                    # check for stop signal
                                                                      31
          for i in range(10):
                                                                                                                                     >aot 0.07775542407106295
13
                                                                                                                                   8 >got 0.5997647474647314
                                                                                     if item is None:
                                                                      32 ~
14
               # generate a value
                                                                                                                                   9 >got 0.7236540952500915
                                                                                                                                   10 >got 0.7956407178426339
                                                                      33
                                                                                         break
               value = random()
15
                                                                                                                                   11 >got 0.11256095725867177
                                                                                                                                   12 Producer: Done
                                                                      34
                                                                                     # report
16
               # block to simulate work
                                                                                                                                   13 >qot 0.9095338767572713
                                                                                     print(f'{time.ctime()} >got {item}')
                                                                      35
                                                                                                                                   14 Consumer: Done
               await asyncio.sleep(value)
17
                                                                                # all done
                                                                      36
18
               # add to the queue
                                                                      37
                                                                                print(f'{time.ctime()} Consumer: Done')
               await queue.put(value)
19
               #print(f'{time.ctime()} Producer: put {value}')
                                                                      38
20
                                                                            # entry point coroutine
          # send an all done signal
                                                                      39
21
          await queue.put(None)
                                                                      40 \vee async def main():
22
          print(f'{time.ctime()} Producer: Done')
                                                                                # create the shared queue
23
                                                                      41
                                                                      42
                                                                                queue = asyncio.Queue()
24
                                                                      43
                                                                                # run the producer and consumers
                                                                                await asyncio.gather(producer(queue), consumer(queue))
                                                                      44
                                                                      45
                                                                            # start the asyncio program
                                                                      46
                                                                            asyncio.run(main())
                                                                      47
```

Example of Asyncio Queue Without Blocking

- This can be achieved by calling the get_nowait() method.
- The get_nowait() function will return immediately. If there is a value in the queue to retrieve, then it is returned.
- Otherwise, if the queue is **empty**, then an **asyncio.QueueEmpty** exception is raised, which can be handled.

```
2-queue-nowait-answer.py > ...
      from random import random
      import asyncio
  3
      import time
  4
  5
      # coroutine to generate work
      async def producer(queue):
 6
          print('Producer: Running')
          # generate work
          for i in range(10):
 9
              # generate a value
 10
              value = random()
 11
              # block to simulate work
 12
              await asyncio.sleep(value)
 13
              # add to the queue
 14
              await queue.put(value)
 15
          # send an all done signal
 16
          await queue.put(None)
 17
          print(f'{time.ctime()} Producer: Done')
 18
 19
```

```
# coroutine to consume work
      asvnc def consumer(queue):
21
           print('Consumer: Running')
22
23
           # consume work
24
           while True:
25
                # get a unit of work without blocking
26
27
                     item = queue.get nowait()
28
                except asvncio.OueueEmptv:
29
                     print(f'{time.ctime()} Consumer: got nothing, waiting a while...')
30
                     await asyncio.sleep(0.5)
31
                     continue
32
                # check for stop
                                                                                     Producer: Runnina
33
                if item is None:
                                                                                     Consumer: Running
                                                                                     Consumer: got nothing, waiting a while...
34
                     break
                                                                                     Consumer: got nothing, waiting a while...
                                                                                     >got 0.896558357626797
35
                # report
                                                                                     Consumer: got nothing, waiting a while...
                                                                                     Consumer: got nothing, waiting a while...
36
                print(f'{time.ctime()} >got {item}')
                                                                                     >qot 0.6498874449486562
37
           # all done
                                                                                     >got 0.14862534743361389
                                                                                  10 Consumer: got nothing, waiting a while...
38
           print(f'{time.ctime()} Consumer: Done')
                                                                                    Consumer: got nothing, waiting a while...
                                                                                  12 >got 0.9271724543351715
39
                                                                                  13 Consumer: got nothing, waiting a while...
                                                                                  14 >aot 0.6659822945662333
      # entry point coroutine
                                                                                  15 >got 0.11205862071348183
                                                                                  16 Consumer: got nothing, waiting a while...
41
      async def main():
                                                                                     Consumer: got nothing, waiting a while...
42
           # create the shared queue
                                                                                  18 >got 0.9490125408623084
                                                                                  19 Consumer: got nothing, waiting a while...
43
           queue = asyncio.Queue()
                                                                                  20 >got 0.150509682492045
                                                                                    >got 0.23281901173320807
44
           # run the producer and consumers
                                                                                  22 Consumer: got nothing, waiting a while...
                                                                                    Consumer: got nothing, waiting a while...
45
           await asyncio.gather(producer(queue), consumer(queue))
                                                                                  24 Producer: Done
46
                                                                                  25 >got 0.8999468879239988
                                                                                  26 Consumer: Done
```

start the asyncio program

asyncio.run(main())

Example of Asyncio Queue With Timeout

- We can get values from the asyncio.Queue by blocking but limited by a timeout
- Instead, we can wrap the get() coroutine in a wait_for() coroutine that supports a timeout. If the timeout elapses before the get() coroutine completes, an asyncio.TimeoutError exception is raised and can be handled

```
♣ 3-queue-waitfor.pv > 分 producer
                                                                    20
                                                                        # consume work
        from random import random
                                                                    21
                                                                         async def consumer(queue):
                                                                    22
                                                                            print(f'{time.ctime()} Consumer: Running')
        import asyncio
                                                                    23
                                                                            # consume work
                                                                    24
                                                                            while True:
        import time
                                                                    25
                                                                                # get a unit of work
  4
                                                                    26
                                                                    27
                                                                                   # retrieve the get() awaitable
        # coroutine to generate work
  5
                                                                    28
                                                                                   get await = queue.get()
        async def producer(queue):
  6
                                                                    29
                                                                                   # await the awaitable with a timeout
                                                                    30
                                                                                   item = await asyncio.wait_for(get_await, 0.5)
             print('Producer: Running')
                                                                    31
                                                                               except asyncio.TimeoutError:
             # generate work
                                                                    32
                                                                                   print(f'{time.ctime()} Consumer: gave up waiting...')
                                                                    33
                                                                                   continue
             for i in range(10):
  9
                                                                    34
                                                                                # check for stop
 10
                  # generate a value
                                                                    35
                                                                                if item is None:
                                                                    36
                                                                                   break
                  value = random()
 11
                                                                    37
                                                                                # report
                  # block to simulate work
                                                                    38
                                                                                print(f'{time.ctime()} >got {item}')
 12
                                                                    39
                                                                            # all done
 13
                  await asyncio.sleep(value)
                                                                    40
                                                                            print('Consumer: Done')
                                                                    41
                  # add to the queue
 14
                                                                    42
                                                                        # entry point coroutine
                   await queue.put(value)
 15
                                                                    43
                                                                         async def main():
                                                                    44
                                                                            # create the shared queue
             # send an all done signal
 16
                                                                    45
                                                                            queue = asyncio.Queue()
             await queue.put(None)
 17
                                                                    46
                                                                            # run the producer and consumers
                                                                            await asyncio.gather(producer(queue), consumer(queue))
             print(f'{time.ctime()} Producer: Done')
 18
 19
                                                                        # start the asyncio program
                                                                        asyncio.run(main())
```

```
>got 0.3050736798012632
  Consumer: gave up waiting...
   >got 0.7019959682053681
  Consumer: gave up waiting...
  >got 0.9753069917130328
12 Consumer: gave up waiting...
  >got 0.7813291071437218
14 Consumer: gave up waiting...
5 >got 0.7831885826899522
16 Consumer: gave up waiting...
  >got 0.8001066750131507
18 Consumer: gave up waiting...
  >got 0.9564293628868409
20 Producer: Done
  >got 0.41507431394001704
22 Consumer: Done
```

Producer: Runnina

Consumer: Running Consumer: gave up waiting...

>got 0.8506665865206575

>got 0.851355213428328

Consumer: gave up waiting...

Example of Asyncio Queue Join and Task Done

- In the previous examples, we have sent a special message (None) into the queue to indicate that all tasks are done.
- An alternative approach is to have coroutines wait on the queue directly and to have the consumer coroutine mark tasks as done.
- This can be achieved via the join() and task_done() functions on the asyncio.Queue.

```
18
                                                                           # coroutine to consume work
♣ 4-queue-task-done.py > 分 consumer
                                                                           async def consumer(queue):
                                                                      19
                                                                       20
                                                                               print(f'{time.ctime()} Consumer: Running')
        from random import random
  1
                                                                       21
                                                                               # consume work
        import asyncio
                                                                       22
                                                                               while True:
                                                                       23
                                                                                   # get a unit of work
        import time
                                                                       24
                                                                                   item = await queue.get()
                                                                       25
                                                                                  # report
                                                                       26
                                                                                   print(f'{time.ctime()} >got {item}')
       # coroutine to generate work
                                                                       27
                                                                                  # block while processing
        async def producer(queue):
                                                                       28
                                                                                   if item:
                                                                       29
                                                                                      await asyncio.sleep(item)
             print(f'{time.ctime()} Producer: Running')
                                                                       30
                                                                                   # mark the task as done
             # generate work
                                                                       31
                                                                                   queue.task done()
                                                                       32
             for i in range(10):
                                                                       33
                                                                           # entry point coroutine
                  # generate a value
 10
                                                                       34
                                                                           asvnc def main():
                  value = random()
                                                                       35
                                                                               # create the shared queue
 11
                                                                       36
                                                                               queue = asyncio.Queue()
 12
                  # block to simulate work
                                                                       37
                                                                               # start the consumer
 13
                  await asyncio.sleep(value)
                                                                       38
                                                                               _ = asyncio.create_task(consumer(queue))
                                                                               # start the producer and wait for it to finish
                                                                       39
                  # add to the queue
 14
                                                                       40
                                                                               await asyncio.create_task(producer(queue))
                                                                               # wait for all items to be processed
 15
                  await queue.put(value)
                                                                       41
                                                                       42
                                                                               await queue.join()
             print(f'{time.ctime()} Producer: Done')
 16
                                                                       43
                                                                           # start the asyncio program
 17
                                                                       44
                                                                           asyncio.run(main())
                                                                       45
```

```
Consumer: Running
Producer: Running
```

>got 0.98439852757525 4 >got 0.31319007221013795

>aot 0.9398085059848861

^{6 &}gt;got 0.14351842921376057

>got 0.24629462902135835 8 >qot 0.4488704344186214

^{9 &}gt;got 0.19476785739518376

^{10 &}gt;got 0.8393990524378161

^{11 &}gt;got 0.3269099694795079

¹² Producer: Done

^{13 &}gt;got 0.8274430954459486

Example of Asyncio Queue With Limited Size

- We can limit the capacity of the queue.
- This can be helpful if we have a large number of producers or slow consumers. It allows us to limit the number of tasks that may be in memory at any one time, limiting the overall memory usage of the application.
- When the queue is full, calls to put() will block until a position becomes available to place another item on the queue.

```
♣ 5-queue-limit-answer.pv > ...
      from random import random
      import asyncio
      import time
      # coroutine to generate work
      async def producer(queue):
          print(f'{time.ctime()} Producer: Running')
          # generate work
          for i in range(10):
 10
              # generate a value
               value = random()
 11
 12
               # block to simulate work
               await asyncio.sleep(value)
 13
              # add to the queue, may block
 14
               await queue.put(value)
 15
          print(f'{time.ctime()} Producer: Done')
 16
 17
```

```
# coroutine to consume work
18
19
     async def consumer(queue):
20
         print(f'{time.ctime()} Consumer: Running')
21
         # consume work
22
         while True:
23
             # get a unit of work
24
             item = await queue.get()
25
             # report
             print(f'{time.ctime()} >got {item}')
26
27
             # block while processing
28
             if item:
29
                  await asyncio.sleep(item)
30
             # mark as completed
             queue.task done()
31
32
         # all done
33
         print(f'{time.ctime()} Consumer: Done')
34
35
     # entry point coroutine
     async def main():
36
         # create the shared queue
37
38
         queue = asyncio.Queue(2)
39
         # start the consumer
40
         = asyncio.create task(consumer(queue))
         # create many producers
41
42
         producers = [producer(queue) for _ in range(5)]
         # run and wait for the producers to finish
43
44
         await asyncio.gather(*producers)
         # wait for the consumer to process all items
45
         await queue.join()
46
47
     # start the asyncio program
48
```

asyncio.run(main())

```
Producer: Running
  Producer: Running
  Producer: Running
  Producer: Running
  Producer: Running
  >got 0.0798149651109541
  >aot 0.5513864113584395
  >got 0.8149184098780632
  >got 0.8561030038666221
  >got 0.8225047439580798
12 >got 0.992630421268497
  >aot 0.27449486943860757
  >got 0.10489939965437134
  >got 0.9004478449122744
  >got 0.9442262069705694
  >got 0.9517905758143422
8 >got 0.38578513687892313
  >got 0.21314357809327322
  >got 0.006412317984848315
  >got 0.522391949578982
  >got 0.4289851852631642
  >got 0.5237185610606917
4 >got 0.7128146789112292
  >got 0.2424277811353306
  >got 0.44543328087703804
  >got 0.36961101864563994
  >got 0.46362053301168127
  >got 0.853341848695711
  >got 0.5234863755930941
  >got 0.04593820030932505
  >got 0.0554357759717663
  >got 0.008185842872241
  >got 0.9700101228192052
  >got 0.8048086100285801
  >got 0.689831779214825
  >got 0.3245915440087028
  >got 0.21373695813973959
  >got 0.9315929425005609
  >got 0.9382045140049264
  >got 0.925811547635268
42 >got 0.6079025826247971
  >got 0.1675603246130124
  >got 0.8861271320774468
  >got 0.5610211824876841
  >got 0.6335242295962565
  Producer: Done
48 >qot 0.5251525663901687
  >got 0.8263850076196841
  >got 0.06117578863178552
  >got 0.7066342593552792
  Producer: Done
  >got 0.883204743564828
  Producer: Done
  >aot 0.06293969547023037
  Producer: Done
  >got 0.5876241223957309
  >got 0.7631673862150006
  Producer: Done
60 >got 0.07354652534254391
61 >got 0.25988256916156316
```