## Лабораторная работа №1

Щетина Полина

Вариант 10

Отчет

[0]**class** *Event*(*mixins*.\_LoopBoundMixin): NEWLINE  
[1]  INDENT"""Asynchronous equivalent to threading.Event.  
 Class implementing event objects. An event manages a flag that can be set  
 to true with the set() method and reset to false with the clear() method.  
 The wait() method blocks until the flag is true. The flag is initially  
 false.  
 """ NEWLINE  
  
[1]**def** \_\_init\_\_(*self*, \*, *loop*=*mixins*.\_marker): NEWLINE  
 [2] INDENT *super*().\_\_init\_\_(*loop=loop*) NEWLINE  
 [2] *self*.\_waiters = *collections*.deque() NEWLINE  
 [2] *self*.\_value = **False** NEWLINE   
[1]**def** \_\_repr\_\_(*self*): NEWLINE  
 [2] INDENT *res* = *super*().\_\_repr\_\_() NEWLINE  
 [2] *extra* = 'set' **if** *self*.\_value **else** 'unset' NEWLINE  
 [2] **if** *self*.\_waiters: NEWLINE  
 [3] INDENT *extra* = *f*'{extra}, waiters:{len(self.\_waiters)}' NEWLINE   
 [2] DEDENT **return** *f*'<{res[1:-1]} [{extra}]>' NEWLINE  
  
[1] **def** *is\_set*(*self*):  
 [2] INDENT """Return True if and only if the internal flag is true.""" NEWLINE  
 [2] return self.\_value NEWLINE  
  
[1] **def** *set(self*):  
 [2] INDENT """Set the internal flag to true. All coroutines waiting for it to become true are awakened. Coroutine that call wait() once the flag is  
 true will not block at all.  
 """ NEWLINE  
 [2] **if not** *self*.\_value: NEWLINE  
 [3] INDENT *self*.\_value = **True** NEWLINE  
 [3] **for** *fut* **in** *self*.\_waiters: NEWLINE  
 [4] INDENT **if not** *fut*.done():NEWLINE  
 [5] INDENT *fut*.set\_*result*(**True**) NEWLINE  
  
[1] **def** *clear*(self): NEWLINE  
 [2] INDENT """Reset the internal flag to false. Subsequently, coroutines calling  
 wait() will block until set() is called to set the internal flag  
 to true again.""" NEWLINE  
 [2] *self*.\_value = **False** NEWLINE  
[1] **async** **def** *wait(self*): NEWLINE  
 [2] INDENT """Block until the internal flag is true. NEWLINE  
 If the internal flag is true on entry, return True  
 immediately. Otherwise, block until another coroutine calls  
 set() to set the flag to true, then return True.  
 """ NEWLINE  
 [2] **if** *self*.\_value: NEWLINE  
 [3]INDENT **return True** NEWLINE  
  
 [2] DEDENT *fut = self*.\_get\_loop().create\_future()NEWLINE  
 [2] *self*.\_waiters.append(*fut*) NEWLINE  
 [2] **try:** NEWLINE  
 [3] INDENT **await** *fut* NEWLINE  
[3] **return True** NEWLINE [2] DEDENT **finally:** NEWLINE  
 [3] INDENT *self*.\_waiters.remove(*fut*) NEWLINE

[0] **class** *Condition(\_ContextManagerMixin, mixins.\_LoopBoundMixin)*: NEWLINE  
[1] INDENT """Asynchronous equivalent to threading.Condition.  
 This class implements condition variable objects. A condition variable  
 allows one or more coroutines to wait until they are notified by another  
 coroutine.  
 A new Lock object is created and used as the underlying lock.  
 """ NEWLINE  
  
[1] **def** \_\_init\_\_(*self, lock*=**None**, \*, *loop=mixins*.\_marker): NEWLINE  
 [2] INDENT *super*().\_\_init\_\_(*loop=loop*) NEWLINE  
 [2] **if** *lock* **is** **None**: NEWLINE  
 [3] INDENT *lock = Lock()*NEWLINE  
  
 [2] DEDENT *self*.\_lock *= loc*k NEWLINE  
 # Export the lock's locked(), acquire() and release() methods. NEWLINE  
[2] *self*.locked = *lock*.locked NEWLINE  
 [2] *self*.acquire = *lock*.acquire NEWLINE  
 [2] *self*.release = *lock*.release NEWLINE  
 [2] self.\_waiters = *collections*.deque()NEWLINE  
  
[1] **def** \_\_repr\_\_(*self*): NEWLINE  
 [2] INDENT *res = super*().\_\_repr\_\_() NEWLINE  
 [2] *extra* = 'locked' **if** *self*.locked() **else** 'unlocked' NEWLINE  
 [2] **if** *self*.\_waiters: NEWLINE  
 [3] INDENT *extra* = *f*'{*extra*}, waiters:{len(*self*.\_waiters)}' NEWLINE  
 [2] DEDENT **return** *f*'<{*res*[1:-1]} [{extra}]>' NEWLINE  
  
[1] **async def** *wait(self*): NEWLINE  
 [2] INDENT """Wait until notified.   
 If the calling coroutine has not acquired the lock when this  
 method is called, a RuntimeError is raised.  
 This method releases the underlying lock, and then blocks  
 until it is awakened by a notify() or notify\_all() call for  
 the same condition variable in another coroutine. Once  
 awakened, it re-acquires the lock and returns True.  
 """ NEWLINE  
[2] **if not** *self*.locked():NEWLINE  
 [3] INDENT **raise** **RuntimeError**('cannot wait on un-acquired lock') NEWLINE  
  
 [2] DEDENT *self*.release()NEWLINE  
[2] **try:** NEWLINE  
 [3] INDENT *fut = self*.\_get\_loop().create\_future()NEWLINE  
 [3] *self*.\_waiters.append(*fut*) NEWLINE  
 [3]**try:** NEWLINE [4] INDENT **await** *fut* NEWLINE[4] **return True** NEWLINE  
 [3] DEDENT **finally**: NEWLINE  
[4] INDENT *self.*\_waiters.remove(*fut*) NEWLINE  
  
 [2] DEDENTDEDENT **finally**: NEWLINE  
 # Must reacquire lock even if wait is cancelled  
 [3] INDENT *cancelled* = **False** NEWLINE  
[3] **while True:** NEWLINE  
 [4] INDENT **try**: NEWLINE  
 [5] INDENT **await** *self*.acquire()NEWLINE  
 [5] **break** NEWLINE [4] DEDENT **except** *exceptions.*CancelledError: NEWLINE  
 [5] INDENT *cancelled* = **True** NEWLINE  
 [4] DEDENT **if** *cancelled*: NEWLINE  
 [5] INDENT **raise** *exceptions.*CancelledError NEWLINE  
  
[1] **async def** *wait\_for(self, predicate):* NEWLINE [2] INDENT """Wait until a predicate becomes true.  
 The predicate should be a callable which result will be  
 interpreted as a boolean value. The final predicate value is  
 the return value.  
 """ NEWLINE  
 [2] *result = predicate()*NEWLINE  
 [2] **while not** *result:* NEWLINE  
 [3] INDENT **await** *self.wait()*NEWLINE[3] *result = predicate()*NEWLINE[2] DEDENT **return** *result* NEWLINE  
[1] **def** *notify(self, n=1):* NEWLINE  
 [2] INDENT """By default, wake up one coroutine waiting on this condition, if any.  
 If the calling coroutine has not acquired the lock when this method  
 is called, a RuntimeError is raised.  
 This method wakes up at most n of the coroutines waiting for the  
 condition variable; it is a no-op if no coroutines are waiting.  
 Note: an awakened coroutine does not actually return from its  
 wait() call until it can reacquire the lock. Since notify() does  
 not release the lock, its caller should.  
 """ NEWLINE  
 [2] **if not** *self.locked():*NEWLINE  
[3] INDENT **raise** **RuntimeError**('cannot notify on un-acquired lock') NEWLINE