Python

Multi

- Str.decode(encoding = 'UTF-8',errors = 'strict’)
- **encoding** – это кодирование, которое будут использоваться.
- **errors** – здесь могут быть даны установки других схем обработки ошибок. Значение по умолчанию для ошибок «strict», это означает, что ошибки кодирования поднимают UnicodeError. Другие возможные значения: ‘ignore’, ‘replace’, ‘xmlcharrefreplace’, ‘backslashreplace’ и любое другое имя, зарегистрированное через codecs.register_error().

Потоки Thread

```python
import threading

from bs4 import BeautifulSoup as BS
import re
import requests


def read(number):

    print(threading.currentThread().getName() + '\n')
    page = requests.get('https://www.djangoproject.com/weblog/?page={}'.format(number))
    print()

if __name__ == '__main__':
    for i in range(5):
        my_thread = threading.Thread(target=read, args=(i,))
        my_thread.start()
```

```
Thread-1

Thread-2

Thread-3

Thread-4

Thread-5
```

```python
import threading

from bs4 import BeautifulSoup as BS
import re
import requests



def read(number):

    print(threading.currentThread().getName() + '\n')
    html = requests.get('https://www.djangoproject.com/weblog/?page={}'.format(number))
    print(html)
    bs_str=BS(html,'lxml')
    #h2=bs_str.find('div',role='main').find('div',{'class':'list-news'}).find('h2').text
    #print(h2)




if __name__ == '__main__':
    for i in range(1,6):
        my_thread = threading.Thread(target=read, args=(i,))
        my_thread.start()

    for i in range(1,6):
        my_thread.join()
```

```
Thread-1

Thread-2

Thread-3
Thread-5


Thread-4

<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
```

- bs_str=BS(html.text, 'html.parser')
- bs_str=BS(html,'lxml')

```
Thread-1

Thread-2

Thread-3

Thread-4

Thread-5

<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
```

Логгирование

```
import threading
import logging

from bs4 import BeautifulSoup as BS
import re
import requests

import time


def get_logger():
    logger = logging.getLogger("threading_example")
    logger.setLevel(logging.DEBUG)

    fh = logging.FileHandler("threading.log")
    fmt = '%(asctime)s - %(threadName)s - %(levelname)s - %(message)s'
    formatter = logging.Formatter(fmt)
    fh.setFormatter(formatter)

    logger.addHandler(fh)
    return logger

def read(number, logger):

    logger.debug('read function executing')
    print(threading.currentThread().getName() + '\n')
    time.sleep(5)
    html = requests.get('https://www.djangoproject.com/weblog/?page={}'.format(number))
    print(html)
    #bs_str=BS(html,'lxml')
    #news1=bs_str.find_all('ul',{'class':'list-news'})[0]
    #news2=news1.find_all('h2')
    #print(news1)
    #print('\n')
    return 0
```

```
if __name__ == '__main__':
    logger = get_logger()
    thread_names = ['Mike', 'George', 'Wanda', 'Dingbat', 'Nina','Djohn']
    for i in range(1,6):
        my_thread = threading.Thread(
            target=read, name=thread_names[i], args=(i,logger))
        my_thread.start()

#    for i in range(1,6):
#        my_thread.join()
```

```
George

Wanda

Dingbat

Nina

Djohn

<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
```

```
2019-01-16 23:31:49,545 - George - DEBUG - read function executing
2019-01-16 23:31:49,546 - Wanda - DEBUG - read function executing
2019-01-16 23:31:49,546 - Dingbat - DEBUG - read function executing
2019-01-16 23:31:49,546 - Nina - DEBUG - read function executing
2019-01-16 23:31:49,547 - Djohn - DEBUG - read function executing
```

Использование наследования

```
class MyThread(threading.Thread):

    def __init__(self, number, logger):
        threading.Thread.__init__(self)
        self.number = number
        self.logger = logger


    def run(self):
        logger.debug('Calling read')
        read(self.number, self.logger)
```

```python
def read(number, logger):

    logger.debug('read function executing')
    print(threading.currentThread().getName() + '\n')
    time.sleep(5)
    html = requests.get('https://www.djangoproject.com/weblog/?page={}'.format(number))
    print(html)
    return 0




if __name__ == '__main__':
    logger = get_logger()
    thread_names = ['Mike', 'George', 'Wanda', 'Dingbat', 'Nina', 'Djohn']
    for i in range(1,6):
        thread = MyThread(i, logger)
        thread.setName(thread_names[i])
        thread.start()
```

```
George

Wanda

Dingbat

Nina

Djohn

<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
```

```
2019-01-16 23:38:20,603 - George - DEBUG - Calling read
2019-01-16 23:38:20,604 - George - DEBUG - read function executing
2019-01-16 23:38:20,604 - Wanda - DEBUG - Calling read
2019-01-16 23:38:20,604 - Wanda - DEBUG - read function executing
2019-01-16 23:38:20,604 - Dingbat - DEBUG - Calling read
2019-01-16 23:38:20,605 - Nina - DEBUG - Calling read
2019-01-16 23:38:20,605 - Dingbat - DEBUG - read function executing
2019-01-16 23:38:20,605 - Djohn - DEBUG - Calling read
2019-01-16 23:38:20,605 - Nina - DEBUG - read function executing
2019-01-16 23:38:20,605 - Djohn - DEBUG - read function executing
```

```python
import threading

total = 0

def update_total(amount):

    global total
    total += amount
    print (total)


if __name__ == '__main__':
    for i in range(10):
        my_thread = threading.Thread(target=update_total, args=(5,))
        my_thread.start()
```

Использование Lock()

- lock.acquire()
- try:
- ... доступ к разделяемому ресурсу
- finally:
- lock.release() # освободить блокировку, что бы ни произошло

```python
import threading

total = 0
lock = threading.Lock()

def update_total(amount):

    global total
    lock.acquire()

    try:
        total += amount
    finally:
        lock.release()

    print (total)


if __name__ == '__main__':
    for i in range(10):
        my_thread = threading.Thread(target=update_total, args=(5,))
        my_thread.start()
```

```
5
10
15
20
25
30
35
40
45
50
```

- lock = Lock()
- lock.acquire() # заблокирует выполнение, если блокировка кем-то захвачена
- ... доступ к разделяемому ресурсу
- lock.release()

```python
import threading

total = 0
lock = threading.Lock()

def update_total(amount):
    
    global total
    with lock:
        total += amount

    print (total)


if __name__ == '__main__':
    for i in range(10):
        my_thread = threading.Thread(target=update_total, args=(5,))
        my_thread.start()
```

```
5
10
15
20
25
30
35
40
45
50
```

- with lock:
- ... доступ к разделяемому ресурсу

```python
import threading

total = 0
lock = threading.Lock()

def do_something():
    lock.acquire()

    try:
        print('Lock acquired in the do_something function')
    finally:
        lock.release()
        print('Lock released in the do_something function')

    return "Done doing something"


def do_something_else():
    lock.acquire()

    try:
        print('Lock acquired in the do_something_else function')
    finally:
        lock.release()
        print('Lock released in the do_something_else function')

    return "Finished something else"


if __name__ == '__main__':
    result_one = do_something()
    result_two = do_something_else()
```

```python
import threading

total = 0
lock = threading.RLock()

def do_something():
    with lock:
        print('Lock acquired in the do_something function')

    print('Lock released in the do_something function')

    return "Done doing something"


def do_something_else():
    with lock:
        print('Lock acquired in the do_something_else function')

    print('Lock released in the do_something_else function')
    return "Finished something else"


def main():
    with lock:
        result_one = do_something()
        result_two = do_something_else()

    print (result_one)
    print (result_two)


if __name__ == '__main__':
    main()
```

```
MacBook-Pro-Dmitrij:mult_new dmitrijstennikov$ python3 thread_09_.py
Lock acquired in the do_something function
Lock released in the do_something function
Lock acquired in the do_something_else function
Lock released in the do_something_else function
MacBook-Pro-Dmitrij:mult_new dmitrijstennikov$ python3 thread_10_.py
Lock acquired in the do_something function
Lock released in the do_something function
Lock acquired in the do_something_else function
Lock released in the do_something_else function
Done doing something
Finished something else
```

lock.acquire(False)

- if not lock.acquire(False):
- ... не удалось заблокировать ресурс
- else:
- try:
- ... доступ к разделяемому ресурсу
- finally:
- lock.release()

acquire может принимать флаг False, если установить флаг в **False**, метод не блокируется, но вернёт **False**, если блокировка кем-то захвачена

lock.locked()

- if not lock.locked():
- # другой поток может начать выполняться перед тем как мы перейдём к следующему оператору
- lock.acquire() # всё равно может заблокировать выполнение

Проблемы Lock()

```
import threading
lock = threading.Lock()
def get_first_part():
    lock.acquire()
    try:
        # берем данные для первой части из общих ресурсах.
    finally:
        lock.release()
    return data
def get_second_part():
    lock.acquire()
    try:
        # берем данные для второй части из общих ресурсах.
    finally:
        lock.release()
    return data
```

```
def get_both_parts():
    first = get_first_part()
    second = get_second_part()
    return first, second
```

Возможное решение

```
def get_both_parts():
    lock.acquire()
    try:
        first = get_first_part()
        second = get_second_part()
    finally:
        lock.release()
    return first, second
```

Решение с RLock()

- import threading
- lock = threading.RLock()
- def get_first_part():
- lock.acquire()
- try:
- # берем данные для первой части из общих ресурсах.
- finally:
- lock.release()
- return data
- def get_second_part():
- lock.acquire()
- try:
- # берем данные для второй части из общих ресурсах.
- finally:
- lock.release()
- return data

RLock()

- lock = threading.Lock()
- lock.acquire()
- lock.acquire() # вызов заблокирует выполнение

- lock = threading.RLock()
- lock.acquire()
- lock.acquire() # вызов не заблокирует выполнение

блокирует поток только в том случае, если блокировка захвачены *другим* потоком, в то время как обычные блокировки могут заблокировать тот же поток, если тот попытается захватить её повторно

BoundedSemaphore()

- semaphore = threading.BoundedSemaphore()
- semaphore.acquire() # уменьшает счетчик
- ... доступ к общему ресурсу
- semaphore.release() # увеличивает счетчик

Внутри семафора - счетчик, в отличии от объекта блокировки, в которой просто флажок. Семафор блокирует поток только когда более заданного числа потоков пытаются захватить семафор.

- max_connections = 10
-
- semaphore = threading.BoundedSemaphore(max_connections)

Event()

- event = threading.Event()
- # поток клиента может подождать пока флажок будет установлен
- event.wait()
- # серверный поток может установить или сбросить флажок
- event.set()
- event.clear()

Condition()

- lock = threading.RLock()
- condition_1 = threading.Condition(lock)
- condition_2 = threading.Condition(lock)

- # Поток производителя
- ... генерация товара
-
- condition.acquire()
- ... добавление товара в ресурс
-
- condition.notify() # отправляем уведомление о новом товаре
- condition.release()

- # Поток потребителя
- condition.acquire()
- while True:
- ... получаем товар из ресурсов
- if item:
- break
- condition.wait() # в противном случае ожидаем поступление товара
- condition.release()
- ... обработка товара

subprocess и Timer

```python
import subprocess

from threading import Timer

kill = lambda process: process.kill()
cmd = ['ping', 'www.google.com']
ping = subprocess.Popen(
    cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

my_timer = Timer(5, kill, [ping])

try:
    my_timer.start()
    stdout, stderr = ping.communicate()
finally:
    my_timer.cancel()
print (str(stdout))
```

```
MacBook-Pro-Dmitrij:mult_new dmitrijstennikov$ python3 thread_11_.py
b'PING www.google.com (173.194.222.104): 56 data bytes\n64 bytes from 173.194.22
2.104: icmp_seq=0 ttl=39 time=36.024 ms\n64 bytes from 173.194.222.104: icmp_seq
=1 ttl=39 time=36.093 ms\n64 bytes from 173.194.222.104: icmp_seq=2 ttl=39 time=
33.621 ms\n64 bytes from 173.194.222.104: icmp_seq=3 ttl=39 time=38.769 ms\n64 b
ytes from 173.194.222.104: icmp_seq=4 ttl=39 time=33.842 ms\n'
MacBook-Pro-Dmitrij:mult_new dmitrijstennikov$
```

Загрузка файлов

```python
import os
from urllib.request import urlopen
from threading import Thread

class DownloadThread(Thread):

    def __init__(self, url, name):
        Thread.__init__(self)
        self.name = name
        self.url = url

    def run(self):
        handle = urlopen(self.url)
        fname = os.path.basename(self.url)

        with open(fname, "wb") as f_handler:
            while True:
                chunk = handle.read(1024)
                if not chunk:
                    break
                f_handler.write(chunk)

        msg = "%s закончил загрузку %s!" % (self.name, self.url)
        print(msg)


def main(urls):
    for item, url in enumerate(urls):
        name = "Поток %s" % (item+1)
        thread = DownloadThread(url, name)
        thread.start()
```

```
Поток 5 закончил загрузку http://www.irs.gov/pub/irs-pdf/f1040sb.pdf!
Поток 1 закончил загрузку http://www.irs.gov/pub/irs-pdf/f1040.pdf!
Поток 4 закончил загрузку http://www.irs.gov/pub/irs-pdf/f1040es.pdf!
Поток 3 закончил загрузку http://www.irs.gov/pub/irs-pdf/f1040ez.pdf!
Поток 2 закончил загрузку http://www.irs.gov/pub/irs-pdf/f1040a.pdf!
MacBook-Pro-Dmitrij:mult_new dmitriistepnikov$
```

Использование очередей

```python
import os
import threading
import urllib.request
from queue import Queue


class Downloader(threading.Thread):
    def __init__(self, queue):
        threading.Thread.__init__(self)
        self.queue = queue

    def run(self):
        while True:
            url = self.queue.get()
            self.download_file(url)
            self.queue.task_done()

            msg = "закончил загрузку %s!" % (self.queue)
            print(msg)

    def download_file(self, url):
        handle = urllib.request.urlopen(url)
        fname = os.path.basename(url)

        with open(fname, "wb") as f:
            while True:
                chunk = handle.read(1024)
                if not chunk:
                    break
                f.write(chunk)

def main(urls):
    queue = Queue()
    for i in range(5):
        t = Downloader(queue)
        t.setDaemon(True)
        t.start()

    for url in urls:
        queue.put(url)
```

Модуль multiprocessing

```python
import os
from multiprocessing import Process


def doubler(number):
    """
    Функция умножитель на два
    """
    result = number * 2
    proc = os.getpid()
    print('{0} doubled to {1} by process id: {2}'.format(
        number, result, proc))


if __name__ == '__main__':
    numbers = [5, 10, 15, 20, 25]
    procs = []

    for index, number in enumerate(numbers):
        proc = Process(target=doubler, args=(number,))
        procs.append(proc)
        proc.start()

    for proc in procs:
        proc.join()
```

```
5 doubled to 10 by process id: 25650
10 doubled to 20 by process id: 25651
15 doubled to 30 by process id: 25652
20 doubled to 40 by process id: 25653
25 doubled to 50 by process id: 25654
```

```python
import os
from multiprocessing import Process, current_process


def doubler(number):
    result = number * 2
    proc_name = current_process().name
    print('{0} doubled to {1} by: {2}'.format(
        number, result, proc_name))


if __name__ == '__main__':
    numbers = [5, 10, 15, 20, 25]
    procs = []
    proc = Process(target=doubler, args=(5,))

    for index, number in enumerate(numbers):
        proc = Process(target=doubler, args=(number,))
        procs.append(proc)
        proc.start()

    proc = Process(target=doubler, name='Test', args=(2,))
    proc.start()
    procs.append(proc)

    for proc in procs:
        proc.join()
```

```
5 doubled to 10 by: Process-2
10 doubled to 20 by: Process-3
15 doubled to 30 by: Process-4
20 doubled to 40 by: Process-5
25 doubled to 50 by: Process-6
2 doubled to 4 by: Test
```

Lock()

```python
from multiprocessing import Process, Lock


def printer(item, lock):
    lock.acquire()
    try:
        print(item)
    finally:
        lock.release()


if __name__ == '__main__':
    lock = Lock()
    items = ['tango', 'foxtrot', 10]

    for item in items:
        p = Process(target=printer, args=(item, lock))
        p.start()
```

```
tango
foxtrot
10
```

logging

```python
import logging
import multiprocessing
from multiprocessing import Process, Lock


def printer(item, lock):

    lock.acquire()
    try:
        print(item)
    finally:
        lock.release()


if __name__ == '__main__':
    lock = Lock()
    items = ['tango', 'foxtrot', 10]
    multiprocessing.log_to_stderr()

    logger = multiprocessing.get_logger()
    logger.setLevel(logging.INFO)

    for item in items:
        p = Process(target=printer, args=(item, lock))
        p.start()
```

```
[INFO/MainProcess] process shutting down
[INFO/Process-1] child process calling self.run()
tango
[INFO/Process-1] process shutting down
[INFO/Process-1] process exiting with exitcode 0
[INFO/MainProcess] calling join() for process Process-3
[INFO/Process-2] child process calling self.run()
foxtrot
[INFO/Process-2] process shutting down
[INFO/Process-2] process exiting with exitcode 0
[INFO/Process-3] child process calling self.run()
10
[INFO/Process-3] process shutting down
[INFO/Process-3] process exiting with exitcode 0
[INFO/MainProcess] calling join() for process Process-1
[INFO/MainProcess] calling join() for process Process-2
```

Pool

```python
from multiprocessing import Pool


def doubler(number):
    return number * 2


if __name__ == '__main__':
    numbers = [5, 10, 20]
    pool = Pool(processes=3)
    print(pool.map(doubler, numbers))
```

```
[10, 20, 40]
```

```python
from multiprocessing import Pool


def doubler(number):
    return number * 2


if __name__ == '__main__':
    pool = Pool(processes=3)
    result = pool.apply_async(doubler, (25,))
    print(result.get(timeout=1))
```

```
50
```

Работа с очередями

```python
from multiprocessing import Process, Queue

sentinel = -1

def creator(data, q):
    print('Creating data and putting it on the queue')
    for item in data:
        q.put(item)


def my_consumer(q):
    while True:
        data = q.get()
        print('data found to be processed: {}'.format(data))

        processed = data * 2
        print(processed)

        if data is sentinel:
            break


if __name__ == '__main__':
    q = Queue()
    data = [5, 10, 13, -1]

    process_one = Process(target=creator, args=(data, q))
    process_two = Process(target=my_consumer, args=(q,))

    process_one.start()
    process_two.start()

    q.close()
    q.join_thread()

    process_one.join()
    process_two.join()
```

```
Creating data and putting it on the
data found to be processed: 5
10
data found to be processed: 10
20
data found to be processed: 13
26
data found to be processed: -1
-2
```