**Библиотека schedule – CRON на Python**

Вам приходилось работать с [CRON](https://ru.wikipedia.org/wiki/Cron)? Это такой сервис в nix-системах, который позволяет регулярно в определенные моменты времени запускать скрипты или программы. Штука с долгой историей, в наследство которой достался странный синтаксис для описания правил:

0 \* \* \* \* my\_script

Что если бы мы хотели иметь свой CRON внутри программы Python, чтобы в нужные моменты времени вызывать функции? Да еще, чтобы у него был человеческий синтаксис? Такая библиотека есть и называется [schedule](https://github.com/dbader/schedule).

pip install schedule

Рассмотрим пример:

**import** schedule

**import** time

**def** job():

print("Работаю")

schedule.every(10).minutes.do(job)

schedule.every().hour.do(job)

schedule.every().day.at("10:30").do(job)

schedule.every(5).to(10).minutes.do(job)

schedule.every().monday.do(job)

schedule.every().wednesday.at("13:15").do(job)

schedule.every().minute.at(":17").do(job)

# нужно иметь свой цикл для запуска планировщика с периодом в 1 секунду:

**while** **True**:

schedule.run\_pending()

time.sleep(1)

Как видите, правила для задания временных интервалов прекрасно читаются, словно они предложения на английском языке. Перевод пары примеров:

# спланируй.каждые(10).минут.сделать(работу)

schedule.every(10).minutes.do(job)

# спланируй.каждый().день.в(10:30).сделать(работу)

schedule.every().day.at("10:30").do(job)

В задания можно передавать параметры вот так:

**def** greet(name):

print('Hello', name)

schedule.every(2).seconds.do(greet, name='Alice')

Если по какой-то причине нужно отменить задание, это делается так:

**def** job1():

# возвращаем такой токен, и это задание снимается с выполнения в будущем

**return** schedule.CancelJob

schedule.every().day.at('22:30').do(job1)

Если нужно отменить группу заданий, то к ним добавляют тэги:

schedule.every().day.do(greet, 'Monica').tag('daily-tasks')

schedule.every().day.do(greet, 'Derek').tag('daily-tasks')

schedule.clear('daily-tasks') # массовая отмена по тэгу

Метод to позволяет задать случайный интервал для выполнения задания, например от 5 до 10 секунд:

schedule.every(5).to(10).seconds.do(my\_job)

Библиотека сама не обрабатывает сама исключения в ваших задачах, поэтому, возможно, понадобится создать подкласс планировщика, как [в этом примере](https://gist.github.com/mplewis/8483f1c24f2d6259aef6). Или декоратор, который будет отменять работу, если произошло исключение. Вот так:

**import** functools

# декоратор для ловли исключений

**def** catch\_exceptions(cancel\_on\_failure=**False**):

**def** catch\_exceptions\_decorator(job\_func):

@functools.wraps(job\_func)

**def** wrapper(\*args, \*\*kwargs):

**try**:

**return** job\_func(\*args, \*\*kwargs)

**except**:

**import** traceback

print(traceback.format\_exc())

**if** cancel\_on\_failure:

**return** schedule.CancelJob

**return** wrapper

**return** catch\_exceptions\_decorator

@catch\_exceptions(cancel\_on\_failure=**True**)

**def** bad\_task():

# даст исключение, но декоратор просто отменит эту задачу

**return** 1 / 0

schedule.every(5).minutes.do(bad\_task)

Если задания занимают продолжительное время или должны выполняться параллельно, то вам самостоятельно придется организовать их выполнение в отдельных потоках.

**import** threading

**import** time

**import** schedule

# код задания

**def** job():

print("Выполняюсь в отдельном потоке")

**def** run\_threaded(job\_func):

job\_thread = threading.Thread(target=job\_func)

job\_thread.start()

schedule.every(10).seconds.do(run\_threaded, job)

schedule.every(10).seconds.do(run\_threaded, job)

schedule.every(10).seconds.do(run\_threaded, job)

schedule.every(10).seconds.do(run\_threaded, job)

schedule.every(10).seconds.do(run\_threaded, job)

# бесконечный цикл, проверяющий каждую секунду, не пора ли запустить задание

**while** 1:

schedule.run\_pending()

time.sleep(1)