Институт цифрового развития
Кафедра инфокоммуникаций

«Управление потоками в Python»

ОТЧЕТ
по лабораторной работе №23
дисциплины
«Основы программной инженерии»

Выполнил:
Мизин Глеб Егорович
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

_____
(подпись)

Проверил:
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Примеры:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread
from time import sleep


def func():
    for i in range(5):
        print(f"from child thread: {i}")
        sleep(0.5)


if __name__ == '__main__':
    th = Thread(target=func)
    th.start()

    for i in range(5):
        print(f"from main thread: {i}")
    sleep(1)
```

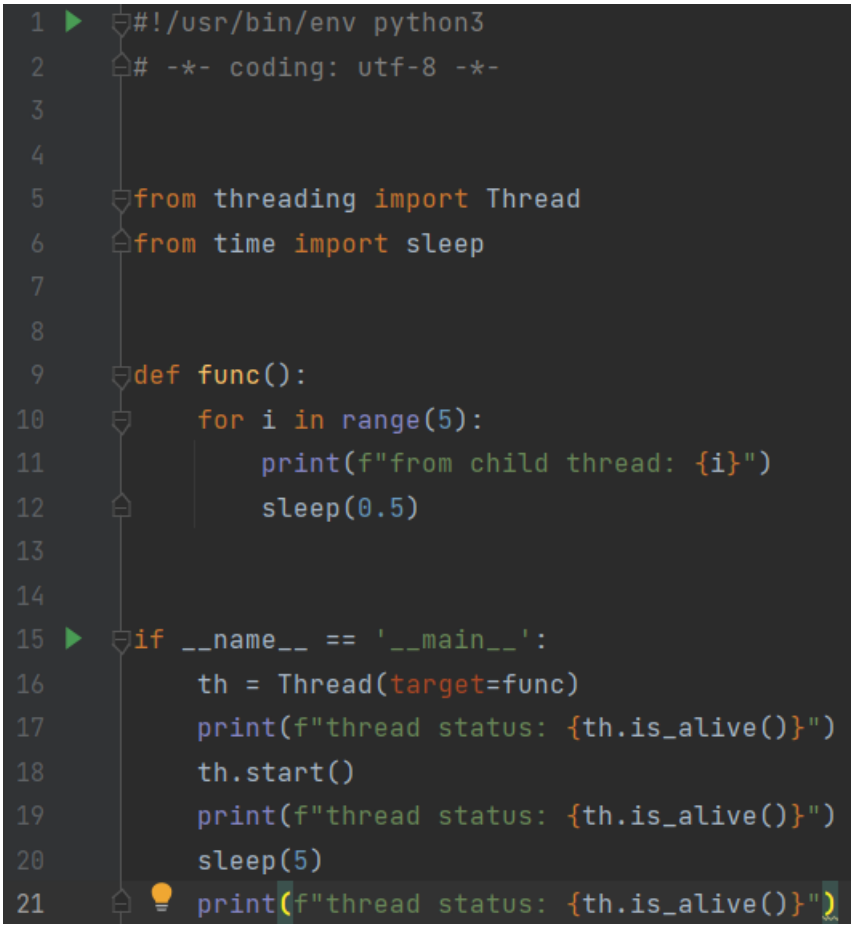
```python
# -*- coding: utf-8 -*-


from threading import Thread
from time import sleep


class CustomThread(Thread):
    def __init__(self, limit):
        Thread.__init__(self)
        self.limit_ = limit

    def run(self):
        for i in range(self.limit_):
            print(f"from CustomThread: {i}")
            sleep(0.5)


if __name__ == '__main__':
    cth = CustomThread(3)
    cth.start()
```



```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread
from time import sleep


def func():
    for i in range(5):
        print(f"from child thread: {i}")
        sleep(0.5)


if __name__ == '__main__':
    th = Thread(target=func)
    print(f"thread status: {th.is_alive()}")
    th.start()
    print(f"thread status: {th.is_alive()}")
    sleep(5)
    print(f"thread status: {th.is_alive()}")
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread, Lock
from time import sleep


lock = Lock()
stop_thread = False


def infinit_worker():
    print("Start infinit_worker()")
    while True:
        print("--> thread work")
        lock.acquire()
        if stop_thread is True:
            break
        lock.release()
        sleep(0.1)
    print("Stop infinit_worker()")


if __name__ == '__main__':
    # Create and start thread
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Thread
from time import sleep


def func():
    for i in range(5):
        print(f"from child thread: {i}")
        sleep(0.5)


if __name__ == '__main__':
    th = Thread(target=func, daemon=True)
    th.start()
    print("App stop")
```

Индивидуальное задание:

$$S = \sum_{n=1}^{\infty} \frac{\cos nx}{n} = \cos x + \frac{\cos 2x}{2} + \ldots; \; x = \pi; \; y = -\ln\left(2\sin\frac{x}{2}\right).$$

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from threading import Thread
from math import cos, pi, log, sin


eps = .0000001


def inf_sum(x):
    summa = x
    prev = 0
    i = 1
    while abs((summa - prev) > eps):
        prev = summa
        summa += (cos(x*i))/i
        i += 1

    print(f"The sum is: {summa}")


def check(x):
    result = -1 * log(2*sin(x/2))

    print(f"The check sum is: {result}")


if __name__ == '__main__':
    x = pi
    thread_1 = Thread(target=inf_sum(x))
    thread_2 = Thread(target=check(x))
    thread_1.start()
    thread_2.start()
```

Контрольные вопросы

1. Что такое синхронность и асинхронность?

Синхронное выполнение программы подразумевает последовательное выполнение операций. Асинхронное – предполагает возможность независимого выполнения задач.

2. Что такое параллелизм и конкурентность?

Конкурентность предполагает выполнение нескольких задач одним исполнителем. Из примера с готовкой: один человек варит картошку и прибирается, при этом, в процессе, он может переключаться: немного прибрался, пошел помешал-посмотрел на картошку, и делает он это до тех пор, пока все не будет готово. Параллельность предполагает параллельное выполнение задач разными исполнителями: один человек занимается готовкой, другой приборкой. В примере с математикой операции и могут выполнять два разных процессора.

3. Что такое GIL? Какое ограничение накладывает GIL?

Конкурентность предполагает выполнение нескольких задач одним исполнителем. Из примера с готовкой: один человек варит картошку и прибирается, при этом, в процессе, он может переключаться: немного прибрался, пошел помешал-посмотрел на картошку, и делает он это до тех пор, пока все не будет готово. Параллельность предполагает параллельное выполнение задач разными исполнителями: один человек занимается готовкой, другой приборкой. В примере с математикой операции и могут выполнять два разных процессора.

4. Каково назначение класса Thread ?

За создание, управление и мониторинг потоков отвечает класс Thread из модуля threading. Поток можно создать на базе функции, либо реализовать свой класс – наследник Thread и переопределить в нем метод run().

5. Как реализовать в одном потоке ожидание завершения другого потока?

Если необходимо дождаться завершения работы потока(ов) перед тем как начать выполнять какую-то другую работу, то воспользуйтесь методом join():

```
th1 = Thread(target=func)
th2 = Thread(target=func)

th1.start()
th2.start()

th1.join()
th2.join()

print("--> stop")
```

У join() есть параметр timeout, через который задается время ожидания завершения работы потоков.

6. Как проверить факт выполнения потоком некоторой работы?

Для того, чтобы определить выполняет ли поток какую-то работу или завершился используется метод *is_alive()*.

```
th = Thread(target=func)
print(f"thread status: {th.is_alive()}")

th.start()
print(f"thread status: {th.is_alive()}")

sleep(5)
print(f"thread status: {th.is_alive()}")
```

7. Как реализовать приостановку выполнения потока на некоторый промежуток времени?

В Python можно использовать функцию time.sleep() для приостановки выполнения потока на некоторый промежуток времени.

```
from threading import Thread
from time import sleep

def func():
    for i in range(5):
        print(f"from child thread: {i}")
        sleep(0.5)

th = Thread(target=func)
th.start()

for i in range(5):
    print(f"from main thread: {i}")
    sleep(1)
```

В приведенном выше примере мы импортировали нужные модули. После этого объявили функцию *func()*, которая выводит пять раз сообщение с числовым маркером с задержкой в 500 мс. Далее создали объект класса *Thread*, в нем, через параметр *target,* указали, какую функцию запускать как поток и запустили его. В главном потоке добавили код вывода сообщений с интервалом в 1000 мс.

8. Как реализовать принудительное завершение потока?

В Python у объектов класса Thread нет методов для принудительного завершения работы потока. Один из вариантов решения этой задачи – это создать специальный флаг, через который потоку будет передаваться сигнал остановки. Доступ к такому флагу должен управляться объектом синхронизации.

```
from threading import Thread, Lock
from time import sleep

lock = Lock()

stop_thread = False

def infinit_worker():
    print("Start infinit_worker()")
    while True:
        print("--> thread work")
        lock.acquire()

        if stop_thread is True:
           break

        lock.release()
        sleep(0.1)

    print("Stop infinit_worker()")

# Create and start thread
th = Thread(target=infinit_worker)
th.start()

sleep(2)

# Stop thread
lock.acquire()
stop_thread = True
lock.release()
```

9. Что такое потоки-демоны? Как создать поток-демон?

Есть такая разновидность потоков, которые называются демоны (терминология взята из мира Unix-подобных систем). Python-приложение не будет закрыто до тех пор, пока в нем работает хотя бы один недемонический поток.

```
def func():
    for i in range(5):
        print(f"from child thread: {i}")
        sleep(0.5)

th = Thread(target=func)
th.start()

print("App stop")
```

Вывод программы:

```
from child thread: 0
App stop
from child thread: 1
from child thread: 2
from child thread: 3
from child thread: 4
```

Как вы можете видеть, приложение продолжает работать, даже после того, как главный поток завершился (сообщение: *"App stop"*).

Для того, чтобы потоки не мешали остановке приложения (т.е. чтобы они останавливались вместе с завершением работы программы) необходимо при создании объекта *Thread* аргументу *daemon* присвоить значение *True*, либо после создания потока, перед его запуском присвоить свойству *deamon* значение *True*. Изменим процесс создания потока в приведенной выше программе:

```
th = Thread(target=func, daemon=True)
```

Запустим ее, получим следующий результат:

```
from child thread: 0
App stop
```

Поток остановился вместе с остановкой приложения.