

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ**

**Отчет о лабораторной работе №11 по дисциплине основы программной
инженерии**

Выполнил:
Выходцев Егор Дмитриевич,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2022 г

1. Примеры из методических указаний

```
e1.py x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4
5      from threading import Thread
6      from time import sleep
7
8
9      def func():
10         for i in range(5):
11             print(f"from child thread: {i}")
12             sleep(0.5)
13
14
15  ▶  if __name__ == '__main__':
16         th = Thread(target=func)
17         th.start()
18         |
19         for i in range(5):
20             print(f"from main thread: {i}")
21             sleep(1)
22
```

```
from child thread: 0
from main thread: 0
from main thread: 1
from main thread: 2
from main thread: 3
from main thread: 4
from child thread: 1
from child thread: 2
from child thread: 3
from child thread: 4

Process finished with exit code 0
```

```
e2.py x
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 from threading import Thread
6 from time import sleep
7
8
9 def func():
10     for i in range(5):
11         print(f"from child thread: {i}")
12         sleep(0.5)
13
14
15 ▶ if __name__ == '__main__':
16     th1 = Thread(target=func)
17     th2 = Thread(target=func)
18     th1.start()
19     th2.start()
20     th1.join()
21     th2.join()
22     print("--> stop")
23
```

```
e2 x
↑ from child thread: 0
↓ from child thread: 1
⏸ from child thread: 1
⏸ from child thread: 2
⏸ from child thread: 2
⏸ from child thread: 3
⏸ from child thread: 3
⏸ from child thread: 4
⏸ from child thread: 4
--> stop

Process finished with exit code 0
```

```
e2.py x
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 from threading import Thread
6 from time import sleep
7
8
9 def func():
10     for i in range(5):
11         print(f"from child thread: {i}")
12         sleep(0.5)
13
14
15 ▶ if __name__ == '__main__':
16     th = Thread(target=func)
17     print(f"thread status: {th.is_alive()}")
18     th.start()
19     print(f"thread status: {th.is_alive()}")
20     sleep(5)
21     print(f"thread status: {th.is_alive()}")
22
```

```
e2 x
C:\Users\student-09-525\PycharmProj
thread status: False
thread status: True
from child thread: 0
from child thread: 1
from child thread: 2
from child thread: 3
from child thread: 4
thread status: False

Process finished with exit code 0
```

```
e2.py x e3.py x
2 # -*- coding: utf-8 -*-
3
4
5 from threading import Thread
6 from time import sleep
7
8
9 class CustomThread(Thread):
10     def __init__(self, limit):
11         Thread.__init__(self)
12         self.limit_ = limit
13
14     def run(self):
15         for i in range(self.limit_):
16             print(f"from CustomThread: {i}")
17             sleep(0.5)
18
19
20 if __name__ == '__main__':
21     cth = CustomThread(3)
22     cth.start()
23
```

```
e3 x
C:\Users\student-09-525\PycharmProje
from CustomThread: 0
from CustomThread: 1
from CustomThread: 2
Process finished with exit code 0
```

```
e2.py × e3.py × e4.py ×
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 from threading import Thread, Lock
6 from time import sleep
7
8
9 lock = Lock()
10 stop_thread = False
11
12
13 def infinit_worker():
14     print("Start infinit_worker()")
15     while True:
16         print("--> thread work")
17         lock.acquire()
18         if stop_thread is True:
19             break
20         lock.release()
21         sleep(0.1)
22     print("Stop infinit_worker()")
23
24
25 ▶ if __name__ == '__main__':
26     # Create and start thread
27     th = Thread(target=infinit_worker)
28     th.start()
29     sleep(2)
30     # Stop thread
31     lock.acquire()
32     stop_thread = True
33     lock.release()
34
```

```
C:\Users\student-09-525\PycharmProje  
Start infinit_worker()  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
--> thread work  
Stop infinit_worker()  
  
Process finished with exit code 0
```

```
e5.py x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4
5      from threading import Thread
6      from time import sleep
7
8
9      def func():
10         for i in range(5):
11             print(f"from child thread: {i}")
12             sleep(0.5)
13
14         |
15  ▶  if __name__ == '__main__':
16         th = Thread(target=func, daemon=True)
17         th.start()
18         print("App stop")
19
```

```
e5 x
C:\Users\student-09-525\PycharmProj
from child thread: 0App stop
Process finished with exit code 0
```

2. Индивидуальное задание (рис 1-2).

С использованием многопоточности для заданного значения x найти сумму ряда S с точностью члена ряда по абсолютному значению $\text{eps} = 10^{-7}$ и произвести сравнение полученной суммы с контрольным значением функции y для двух бесконечных рядов. Номера вариантов необходимо уточнить у преподавателя:

Вариант 5

$$S = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots; \quad x = 0, 3; \quad y = \cos x.$$

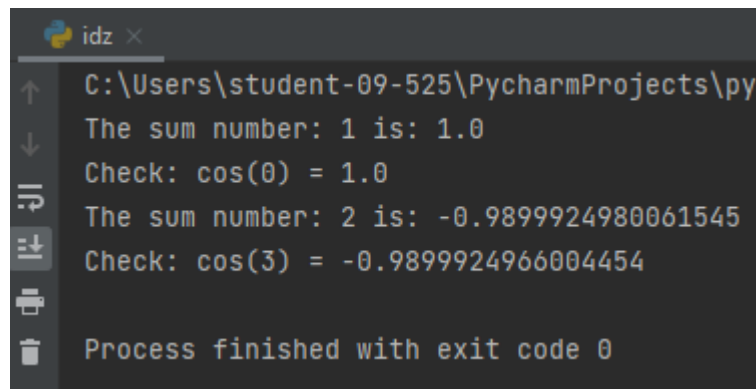
5.


```

1  ▶  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  from threading import Thread
6  from math import factorial, cos
7
8
9  eps = .0000001
10
11
12  def inf_sum(x, check, num):
13      summa = 1
14      i = 1
15      prev = 0
16      while abs(summa - prev) > eps:
17          prev = summa
18          summa += ((-1)**i * (x**(2*i))) / factorial(2*i)
19          #print(summa)
20          i += 1
21      print(f"The sum number: {num} is: {summa}")
22      print(f"Check: cos({x}) = {check}")
23
24
25  ▶  if __name__ == '__main__':
26      checksum1 = cos(0)
27      ⚡ thread1 = Thread(target=inf_sum, args=(0, checksum1, 1))
28      thread1.start()
29      checksum2 = cos(3)
30      thread2 = Thread(target=inf_sum, args=(3, checksum2, 2))
31      thread2.start()
32

```

Рисунок 1 – Код программы



```
idz x
C:\Users\student-09-525\PycharmProjects\py
The sum number: 1 is: 1.0
Check: cos(0) = 1.0
The sum number: 2 is: -0.9899924980061545
Check: cos(3) = -0.9899924966004454

Process finished with exit code 0
```

Рисунок 2 – Результат выполнения многопоточной программы

3. Ответы на 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. Параллельность предполагает параллельное выполнение задач разными исполнителями.

3. Что такое GIL? Какое ограничение накладывает GIL?

GIL — это аббревиатура от Global Interpreter Lock – глобальная блокировка интерпретатора. Он является элементом эталонной реализации языка Python, которая носит название CPython. Суть GIL заключается в том, что выполнять байт код может только один поток. Это нужно для того, чтобы упростить работу с памятью (на уровне интерпретатора) и сделать комфортной разработку модулей на языке C. Это приводит к некоторым особенностям, о которых необходимо помнить. Условно, все задачи можно разделить на две большие группы: в первую входят те, что преимущественно используют процессор для своего выполнения, например, математические, их ещё называют CPU-bound, во вторую – задачи работающие с вводом выводом (диск, сеть и т.п.), такие задачи называют IO-bound. Если вы запустили в одном интерпретаторе несколько потоков, которые в основном используют процессор, то скорее всего получите общее замедление работы, а не прирост производительности. Пока выполняется одна задача, остальные простаивают (из-за GIL), переключение происходит через определенные промежутки времени. Таким образом, в каждый конкретный момент времени, будет выполняться только один поток, несмотря на то, что у вас может быть

многоядерный процессор (или многопроцессорный сервер), плюс ко всему, будет тратиться время на переключение между задачами. Если код в потоках в основном выполняет операции ввода-вывода, то в этом случае ситуация будет в вашу пользу. В CPython все стандартные библиотечные функции, которые выполняют блокирующий ввод-вывод, освобождают GIL, это дает возможность поработать другим потокам, пока ожидается ответ от ОС.

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(). 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().

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

С помощью метода sleep() из модуля time.

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.

```
lock.acquire()
```

```
if stop_thread is True:
```

```
break
```

```
lock.release()
```

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

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 daemon значение True.

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