Пример из медицинской отрасли, потому что в этой отрасли вы часто можете столкнуться с такой ситуацией. Зачастую в клиниках требуются высокие позиции постов по ключевым фразам, при этом на первый план выходят только записи в блогах.

В этой ситуации, даже вкладывая гораздо больший бюджет в позиционирование страницы, чем упомянутые блоги в продвижении своих записей, клиника не сможет достичь более высоких позиций. Единственное решение в этой ситуации, которое позволяет клинике занимать лидирующие позиции по заданной фразе, - это создать запись в блоге, которая будет соответствовать намерениям пользователей, лучше понимаемым Google.

1. Как работает поисковая система Google <https://serpstat.com/ru/blog/kak-ustroen-poisk-google/> (дата обращения: 15.10.2020).
2. Как работает Google Поиск <https://www.google.com/intl/ru/search/howsearchworks/> (дата обращения: 15.10.2020).

Шавтикова Л.М., Текеев М.Б.

Параллельное вычисление потоков на языке программирования Python

*Северо-Кавказская государственная академия
(Россия, Черкесск)*

doi: 10.18411/lj-12-2020-46

idsp: ljjournal-12-2020-46

Аннотация

В статье рассматривается организация параллельных вычислений потоков на языке программирования Python. Представлено выполнение функции в разных классах одновременно.

Ключевые слова: потоки, программирование, параллелизм операций.

Abstract

This article discusses the organization of parallel thread calculations in the Python programming language. Presents the execution of a function in different classes at the same time.

Keywords: threads, programming, and operation parallelism.

Потоки позволяют запустить выполнение нескольких задач в конкурентном режиме в рамках одного процесса интерпретатора. При этом, нужно помнить о GIL. Все потоки будут выполняться на одном CPU, даже если задачи могут выполняться параллельно. Поэтому есть такое правило, если задачи в основном потребляют ресурсы процессора, то следует использовать процессы, если ввод-вывод, то потоки и другие инструменты асинхронного программирования, которые в Python обладают довольно мощным функционалом.

Синхронное - это выполнение программы подразумевает последовательное выполнение операций.

Асинхронное – предполагает возможность независимого выполнения задач.

Для синхронизации доступа к ресурсам из нескольких потоков Python предоставляет набор объектов, каждый из которых обладает рядом особенностей, делающих их пригодными для решения некоторой группы специфических задач.

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 (). Для начала рассмотрим вариант создания потока на базе функции:

```
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 импортированы нужные модули. После этого объявлена функция `func()`, которая выводит пять раз сообщение с числовым маркером с задержкой в 500 мс. Далее создан объект класса `Thread`, в нем, через параметр `target`, указано, какую функцию запускать как поток и был запущен. В главном потоке добавлен код вывода сообщений с интервалом в 1000 мс.

В результате запуска этого кода получилось следующее:

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

Можно увидеть, что код из главного и дочернего потоков выполняются псевдопараллельно (во всяком случае создается такое ощущение), т.к. задержка в дочернем потоке меньше, то сообщение из него появляются чаще.

Если необходимо дождаться завершения работы потока(ов) перед тем как начать выполнять какую-то другую работу, используется метод `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. 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()}")
```

В результате получим следующее:

```
thread status: False
from child thread: 0
thread status: True
from child thread: 1
from child thread: 2
from child thread: 3
from child thread: 4
thread status: False
```

Создание классов наследников от Thread

Ещё один способ создавать и управлять потоками – это реализовать класс наследник от Thread и переопределить у него метод run().

```
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)
cth = CustomThread(3)
cth.start()
```

В терминале получим следующее:

```
from CustomThread: 0
from CustomThread: 1
from CustomThread: 2
```

Потоки-демоны

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.

Пример:

```
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
```

Здесь можно увидеть, что 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 *daemon* 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.

1. Бриггс, Джейсон Python для детей. Самоучитель по программированию / Джейсон Бриггс. - Москва: Огни, 2013. - 177 с.
2. Бэрри, Пол Изучаем программирование на Python / Пол Бэрри. - М.: Эксмо, 2016. - 332 с.
3. Васильев, А. Н. Python на примерах. Практический курс по программированию / А.Н. Васильев. - М.: Наука и техника, 2016. - 432 с.
4. Васильев, Александр Николаевич Python на примерах. Практический курс по программированию. Руководство / Васильев Александр Николаевич. - М.: Наука и техника, 2017. - 752 с.
5. Гуриков, С.Р. Основы алгоритмизации и программирования на Python / С.Р. Гуриков. - М.: Форум, 2018. - 991 с.
6. Гуриков, С.Р. Основы алгоритмизации и программирования на Python. Учебное пособие. Гриф МО РФ / С.Р. Гуриков. - М.: Инфра-М, Форум, 2018. - 707 с.

Шавтикова Л.М., Текеев М.Б.

Создание интерфейса к задаче вычислительной математики с использованием графики

*Северо-Кавказская государственная академия
(Россия, Черкесск)*

doi: 10.18411/lj-12-2020-47

idsp: ljournal-12-2020-47

Аннотация

В статье рассматривается создание интерфейса к задаче вычислительной математики с использованием библиотек для рисования графики на языке программирования Python.

Ключевые слова: программирование, 2D-графика, численный анализ, фреймворк, GUI приложение.

Abstract

The article discusses the creation of an interface to a computational mathematics problem using libraries for drawing graphics in the Python programming language.

Keywords: programming, 2D graphics, numerical analysis, framework, GUI application.

В Python есть довольно много GUI фреймворков (graphical user interface), однако только Tkinter встроен в стандартную библиотеку языка. У Tkinter есть несколько преимуществ. Он кроссплатформенный, поэтому один и тот же код можно использовать на Windows, macOS и Linux. Визуальные элементы отображаются через